A Method for the Generation of HDL Code at the RTL level from a High-Level Formal Specification Language

Apostolos A. Kountouris, Christophe Wolinski
IRISA
Campus Universitaire de Beaulieu
F-35042 Rennes CEDEX
FRANCE

Abstract--In this paper a method for generating HDL code from SIGNAL formal specifications, is described. Applying two transformations on the initial specification yields functionally equivalent RTL HDL code. The functional equivalence is formally proven. The methodology allows component re-usability and enables the validation of their integration at the specification level. We anticipate that the principles presented in this paper, will be applied in the framework of a cooperation with Motorola.

I. Introduction

In this paper we describe a new methodology for the generation of *Hardware Description Language* (HDL) code from a high-level formal specification language. The novelty of this methodology is the use of the formal language semantics to prove the functional equivalence between the initial and the resulting HDL representations. To obtain a viable HDL generation scheme two important issues have to be addressed. Primarily the functional equivalence between specification and implementation and secondly, re-usability. The first issue is the classical problem of guaranteeing that the implementation actually implements the specification. Whenever there is a passage from one representation, usually a high-level one, to another lower level one, there can be discrepancies attributed to the fact of changing representation domains. In addition, being able to re-use past designs is necessary in order to shorten the design cycle by re-using previous work or component libraries. Finally, the generation process should be able to accommodate different HDL target languages and should be guided by a set of user supplied options.

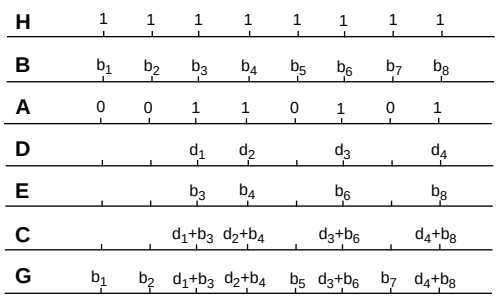
We investigate the benefits of a circuit specification environment where the SIGNAL language is the specification *front-end*. SIGNAL is a dataflow oriented language [1] based on the *synchrony* hypothesis[2]. Using the language expressions the user is programming in an equational style and thus each program is a system of equations. The SIGNAL compiler resolves these systems and checks for non-determinism, circular control/data dependencies, SIGNAL clock constraints (see [1][4]) etc. These checks are useful in discovering possible sources of error, at compile time. In the example shown in Fig. 1a a small SIGNAL program captures the desired functional requirements. In Fig. 1b we can see its *chronogram* that shows how the outputs of the program are produced in response to the inputs and the relative presence of the program signals. For example the value of the output G depends on the value of B if A is *false* or on the value of C if A is *true*. In the *chronogram* we can also see that D caries a value only at the logical instants that A is *true*.

II. HDL Generation Process

The process consists of applying a series of transformations to the initial SIGNAL program. In this way we obtain a SIGNAL program functionally equivalent to the initial one. This program can be directly mapped to a *Register Transfer Leve*l (RTL) HDL representation by applying a third transformation that performs the translation to HDL. In doing this we use the

```
process EXAMPLE1=
     { ? event H;
         logical A;
         integer B, D
       ! integer C, G }
     (| E := B when A
      | G := C default B
      | C := D+E
      | synchro { A, H, B }
      |)
     where
          integer E
end
```

(a) SIGNAL program



(b) temporal and value relationships of the inputs and outputs of the program

Fig. 1. Functional specification in SIGNAL

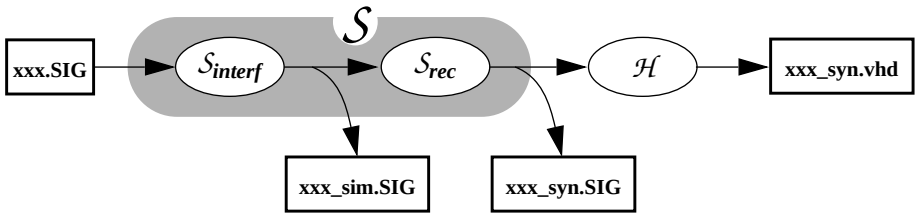


Fig. 2. HDL code generation by successive application of the $\mathcal{S}$ and $\mathcal{H}$ transformations

structural information contained in the SIGNAL description as well as a set of SIGNAL to HDL translation rules, depending on the targeted HDL.

A. The Transformation Process

The three transformations, $\mathcal{S}_{interf}$, $\mathcal{S}_{rec}$, $\mathcal{H}$, are the basic tools used in the HDL generation process. In Fig. 2 we graphically depict the process of generating HDL code (xxx_syn.vhd) from an initial SIGNAL program (xxx.SIG). We formally prove that by applying these transformations on SIGNAL programs we end up with functionally equivalent HDL representations. It is also proven that such a scheme allows the seamless integration of predefined components. Using the SIGNAL formal operational semantics we demonstrate that for a SIGNAL process P in a context C $P \equiv_{\text{SIG}} \mathcal{S}_{interf}(P)$ where $\equiv_{\text{SIG}}$ is our notion of functional equivalence between SIGNAL processes. Next we prove that for a SIGNAL process P that is equal to the parallel composition of processes P_1 and P_2 (i.e. $P=P_1|P_2$),

$$P \equiv_{\text{SIG}} (\mathcal{S}_{interf}(P_1) \mid \mathcal{S}_{interf}(P_2)) .$$

This is in turn used to prove that $\mathcal{S}_{interf}$ can be applied recursively (noted $\mathcal{S}_{rec}$) on the process hierarchy till the level of SIGNAL kernel operators is reached. Consequently $P \equiv_{\text{SIG}} \mathcal{S}_{rec}(P)$. It is shown that for a process P, its $\mathcal{S}_{rec}(P)$ and $\mathcal{H}(\mathcal{S}_{rec}(P))$ (the HDL representation) are observationally equivalent

$$\mathcal{S}_{rec}(P) \equiv_{\text{OBS}} \mathcal{H}(\mathcal{S}_{rec}(P))$$

B. Component Re-use

Finally, we can deduce that we can effectively use libraries of predefined elements. Let LC a library component with LC_{SIG} its description in SIGNAL and LC_{HDL} its description in HDL. If LC is used in a SIGNAL program P, we may represent this fact as $P = P_{sur} \mid LC_{SIG}$ where P_{sur} is the part of P that surrounds LC. If C is the context defined by P, we obtain

$$P \equiv_{\text{SIG}} \mathcal{S}_{rec}(P_{sur}) \mid \mathcal{S}_{interf}(LC_{SIG})$$

In the SIGNAL source code LC is marked as being a library element, $\mathcal{S}_{rec}$ is not applied on it. Such a process is entirely substituted by LC_{HDL}, the pre-existing HDL code (found in the library), during the application of $\mathcal{H}$ that generates the HDL code for P.

III. Specification-Level Optimizations

Using the SIGNAL clock exclusivity information we have many possibilities for specification level optimizations. As a representative example we demonstrate how we can effectively reduce the resources needed and thus minimize hardware. If for instance a SIGNAL program contains an expression like:

d := ((a+b) *when* ev) *default* (b+c)

after clock calculus the clocks of the two + operators are found *mutually exclusive* and thus a single + resource is needed. Fig. 3 (a) shows the $\mathcal{S}$ transformation of the example when clock exclusivity information is not considered. Resource sharing is shown in(b) which is equivalent to (a).
The table below compares (a) and (b) in terms of number of gates and occupied area for 32-bit quantities for a Xilinx XC4000 FPGA implementation [7].

TABLE I
Area Minimization by Resource Sharing

FIFO design	no resource sharing	resource sharing
number of primitive cells	631	480
area	204.5	160.0

IV. Methodology Outline

From a methodology perspective the major steps that lead from the high-level SIGNAL specification of a circuit, to its implementation in hardware, are:
1. *Specification:* Design entry, static validation

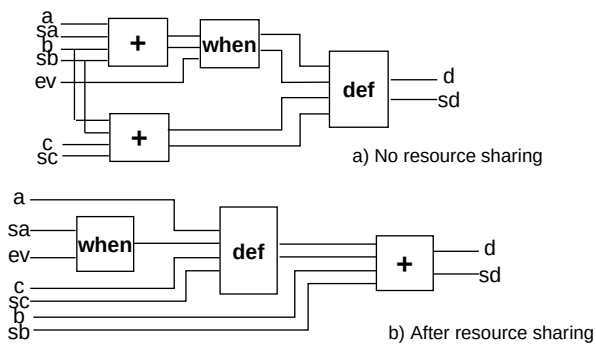


Fig. 3. Resource sharing optimization example.

2. *Synthesizability Analysis:* Apply $\mathcal{S}_{interf}$, static validation of $\mathcal{S}_{interf}$
3. *Dynamic Validation:* software simulation
4. *HDL Generation:* Apply $\mathcal{S}_{rec}$, $\mathcal{H}$
5. *Synthesis:* set constraints, optimization options etc.
6. *Post-synthesis Validation:* gate-level simulation, etc.

This methodology aspect is graphically depicted in Fig. 4. For design entry we use the SIGNAL language either in textual or graphical form. The static validation is performed by the SIGNAL compiler. The passage from SIGNAL to HDL is a multi-stage, sequential, iterative process. Till before step 4 we are always in the SIGNAL domain, meaning that at the end of any preceding step we may use other SIGNAL tools for purposes like property verification etc.

V. Example: HW Implementation of a FIFO

The described HDL generation methodology was applied to a FIFO specification in SIGNAL. Fig. 5 shows the FIFO signal specification and Fig. 6 shows the results at the end of step 3. In the table below we compare the synthesis results for two similar FIFO designs, in terms of number of cells and occupied area. The first is the low area implementation of a synchronous FIFO, found in the SYNOPSYS DesignWare library [6]. The second is the result of our HDL generation process for the SIGNAL specification of Fig. 5. We used the SYNOPSYS *design_analyzer* and the Xilinx FPGA compiler for the Xilinx XC4000 part [7]. The slight differences between the two designs are quite encouraging as far as the quality of the HDL generated by our process is concerned. We expect to further ameliorate these results by a more extensive application of specification level optimizations.

TABLE II
Area Comparison Against a Predefined Component

FIFO design	DesignWare Low Area	SIGNAL
Combinational	204	189.5
Noncombinational	10	63.0
Total area	214	252.5
Number of primitive cells	611	631

VI. Conclusion

In this paper we presented an HDL generation process that accepts as input a SIGNAL program and produces as output a functionally equivalent representation in HDL a the RTL level. To achieve this we use structural information in the SIGNAL program. The most important aspect, of this HDL generation process, is that it constitutes a *formal link* between the specification (SIGNAL) domain and the implementation (synthesis) domain. The whole process is proved using the formal SIG-

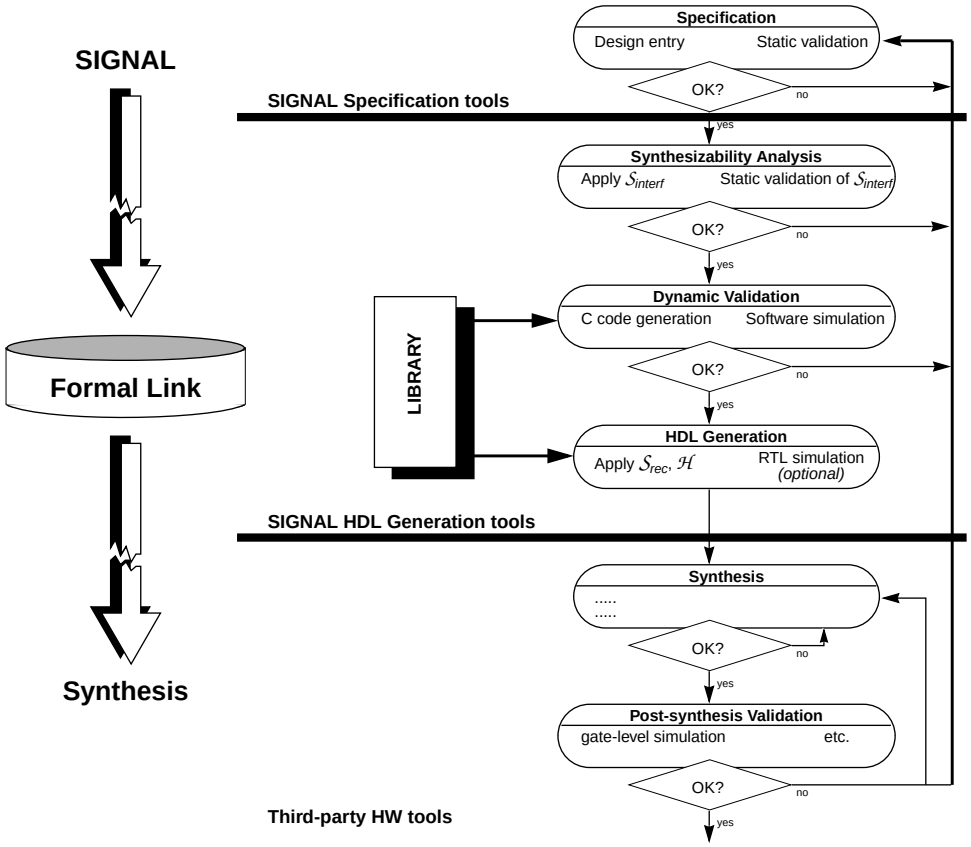


Fig. 4. Methodology steps.

NAL semantics. This fact renders this generation process an acceptable and viable link between specification and implementation. Finally a specification level optimization process was studied with a two-fold intention: first express optimizations in the specification level proving the functional equivalence of the optimized circuit, and second enhance the optimization process performed by the underlying synthesis tools.

REFERENCES

[1] Paul Le Guernic, Michel Le Borgne, Thierry Gautier, Claude Le Maire, "Programming Real Time Applications with SIGNAL", *Proceedings of the IEEE*, vol. 79, no.9, pp. 1321-1336, Sep. 1991.

[2] Albert Benveniste, Gerard Berry, "The Synchronous Approach to Reactive and Real-Time Systems", *Proceedings of the IEEE*, vol. 79, no.9, pp. 1270-1282, Sep. 1991.

[3] "Special section: R/T Programming", *Proceedings of the IEEE*, vol. 79, no.9, Sep. 1991.

[4] Loic Besnard, *Compilation de SIGNAL: horloges, dependances, environment*, PhD thesis, University of Rennes I.

[5] James T. O'Connor, "Synchronous vs. Asynchronous Design Methodology for Digital Circuits Requiring Multiple Clock Signals from a Single Clock Source", *Synopsys on-line documentation*, methodology note.

[6] "DesignWare Components Databook", *Synopsys on-line documentation*.

[7] "XACT: Xilinx Synopsys Interface FPGA User Guide", Xilinx, Dec. 1994.

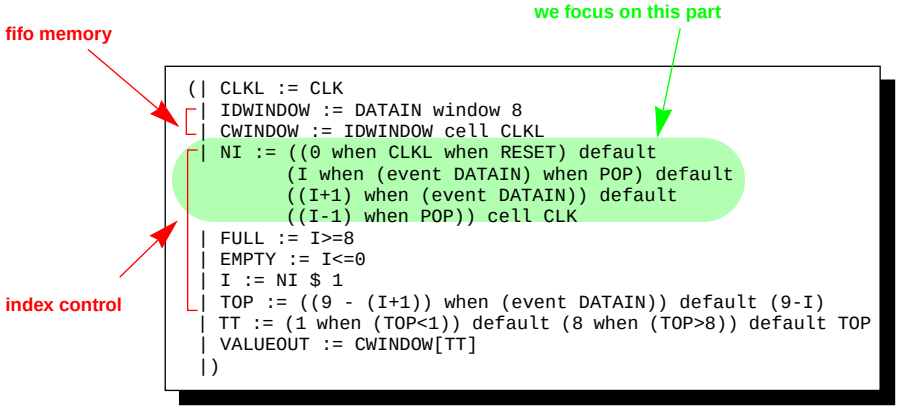


Fig. 5. Specification of the FIFO behavior in SIGNAL

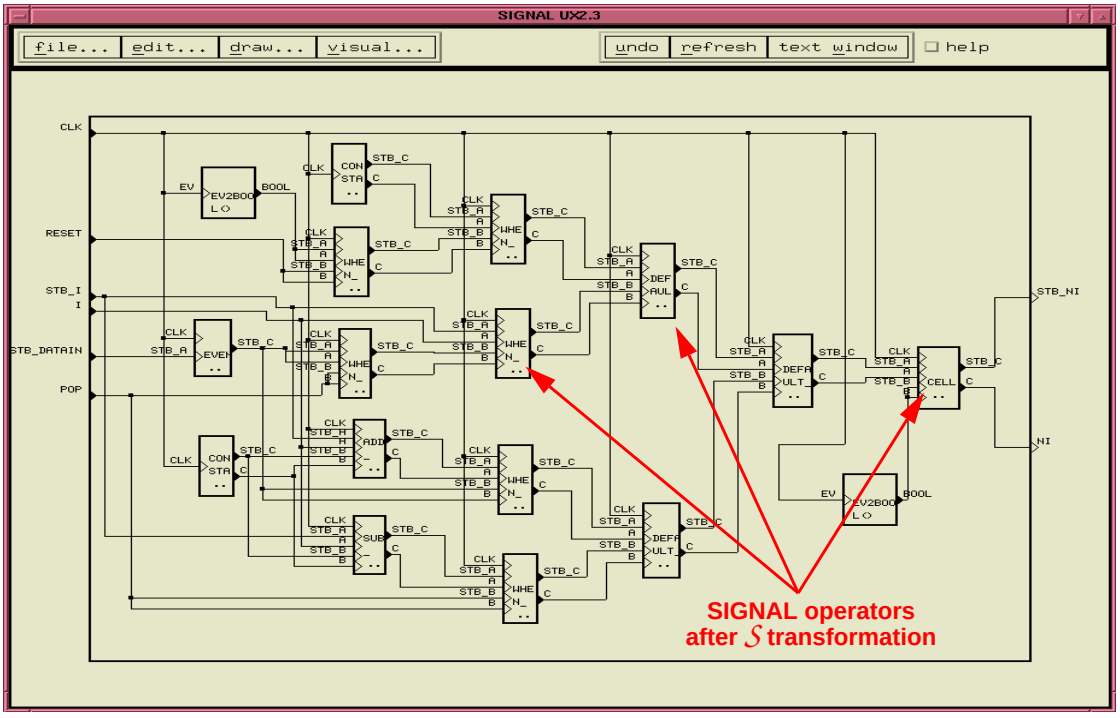


Fig. 6. $\mathcal{S}_{rec}$ transformation (fifo_syn.SIG) before HDL generation for the IOPB FIFO

Technical Track: 6. VLSI and CAD (VLSI circuits, design tools, CAD software)
Format: Lecture

A Method for the Generation of HDL Code at the RTL level from a High-Level Formal Specification Language

Apostolos A. Kountouris, Christophe Wolinski

Abstract--Abstract: In this paper a method for generating HDL code from SIGNAL formal specifications, is described. Applying two transformations on the initial specification yields functionally equivalent RTL HDL code. The functional equivalence is formally proven. The methodology allows component re-usability and enables the validation of their integration at the specification level.

Author Information

Apostolos Kountouris
IRISA
Campus de Beaulieu
F-35042 RENNES CEDEX - FRANCE
E-mail: kountour@irisa.fr
Phone: +33 2 99 84 72 44
Fax: +33 2 99 84 71 71
Telex: UNIRISA 950 473F